Е.И. Сафонов, И.Р. Панчишин
Югорский Государственный Университет, г. Ханты-Мансийск

УДК 004.42

СЕТЕВАЯ МНОГОПЛАТФОРМЕННАЯ ИГРОВАЯ СРЕДА

Аннотация. В статье представлено описание подхода для реализации кроссплатформенной игры «Морской бой» с поддержкой сетевого и одиночного режима

Ключевые слова: Qt, сетевая игра, многоплатформенность, морской бой.

Подходов к программированию приложений со временем становится все больше. Много времени уделяется разработке систем искусственного интеллекта, построению алгоритмов, разнообразие операционных систем требует адаптации программного обеспечения, взаимодействие людей в сети стало необходимостью. Для закрепления знаний в данных областях, было принято решение спроектировать и реализовать сетевую многоплатформенную игровую среду на примере игры «Морской бой».

Мной был выбран язык программирования C++, так как он мультипарадигмальный, кроссплатформенный, и имеет широкий спектр применений: от написания развлекательных приложений до драйверов устройств [1;38]. Средством разработки приложения был выбран кроссплатформенный инструментарий разработки программного обеспечения Qt SDK, который также расширяет возможности языка предварительной системой обработки кода (MOC [2]). В качестве интегрированной среды разработки используется QtCreator. Он отвечает всем современным требованиям [3], а также поддерживает опциональную эмуляцию редактора vim, который я предпочитаю использовать.

Qt позволяет запускать написанное с помощью его ПО на большинстве операционных систем путем простой компиляции программы для каждой ОС без изменения исходного кода. Непосредственное участие в этом выполняет утилита сборки проекта- qmake [4;72].

Графический интерфейс моего приложения был написан с помощью библиотек модуля QtWidgets, для оформления которого была задействована технология рисования Arthur и стили QSS.

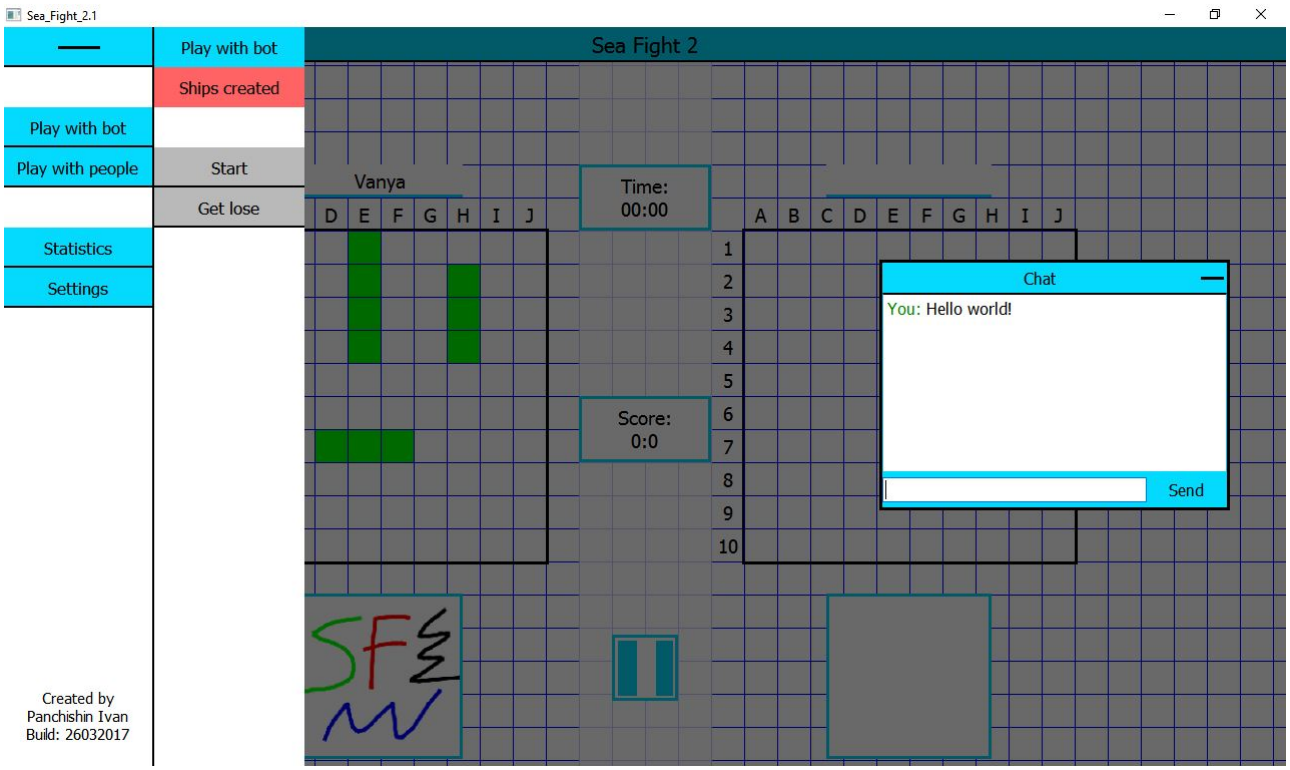

Рис. 1. Графический интерфейс приложения «Морской бой»

Приложение имеет:

1. Игровой таймер, реализованный с помощью QTimer класса.
2. Виджет для рисования, который является полиморфной реализацией QWidget класса с переопределенными специализированными обработчиками событий мыши и событием перерисовки виджета, в котором задействован массив, хранящий точки для рисования.
3. Показатель счета, представленный в виде label виджета, который отображает актуальные данные в зависимости от настоящего соперника. Эти данные хранятся в ассоциативном шаблонном контейнере QMap с привязкой к именам противников.

4. Индикатор, сигнализирующий о «смене хода». Который представляет собой отображение QPixmap объекта класса, разработанного для отображения картинки на экране, на QLabel виджете.

5. Графическую сцену в виде игрового поля (QGraphicsScene), работающую по принципу «интервью» или «модель-представление».

6. Подвижный, сворачиваемый чат, который включает в себя комплексное использование виджетов ввода и отображения благодаря механизму объектной иерархии.

7. Игровое меню, представленное в виде QStackedWidget, которое состоит из:

A) Подменю для начала игры с ботом или сетевой игры.

B) Подменю настроек, где можно установить Nick или изменить размер игрового поля.

C) Подменю, где можно посмотреть актуальную статистику о всех играх.

Взаимодействие с пользовательским интерфейсом во многом построено с помощью сигнал-слотового механизма Qt, а также: механизма событий, перехвата событий с помощью фильтров и использования лямбда выражений, добавленных в стандарте C++11 [5] и Qt5 [6].

Стоит уделить внимание реализации бота для одиночной игры. Основная идея заключается в том, чтобы сделать его действия максимально неотличимыми от действий человека, который не принимает нерациональных действий во время игры.

Алгоритм для определения позиции для выстрела реализован в функции, с возвращаемым типом данных bool. Она возвращает true, если было попадание, и false, если был промах, или выстрел завершился концом игры. Данную функцию можно использовать в качестве аргумента цикла while. Также она выполняет в себе 3 последовательных этапа: поиск первой палубы, второй и последующих.

На первом этапе определяется самый длинный целый корабль пользователя. Исходя из этого из массива всех допустимых номеров ячеек для атаки формируется массив с номерами, в которых целиком может уместиться данный корабль. Поиск потенциально самого большого корабля существенно уменьшит область поиска кораблей размерами меньше в случае успеха. Случайная позиция для выстрела берется из сформированного массива, индекс которого находится случайным образом при помощи функции rand() библиотеки cmath, диапазон возвращаемого числа которой должен быть равен от 0 до максимального количества элементов массива.

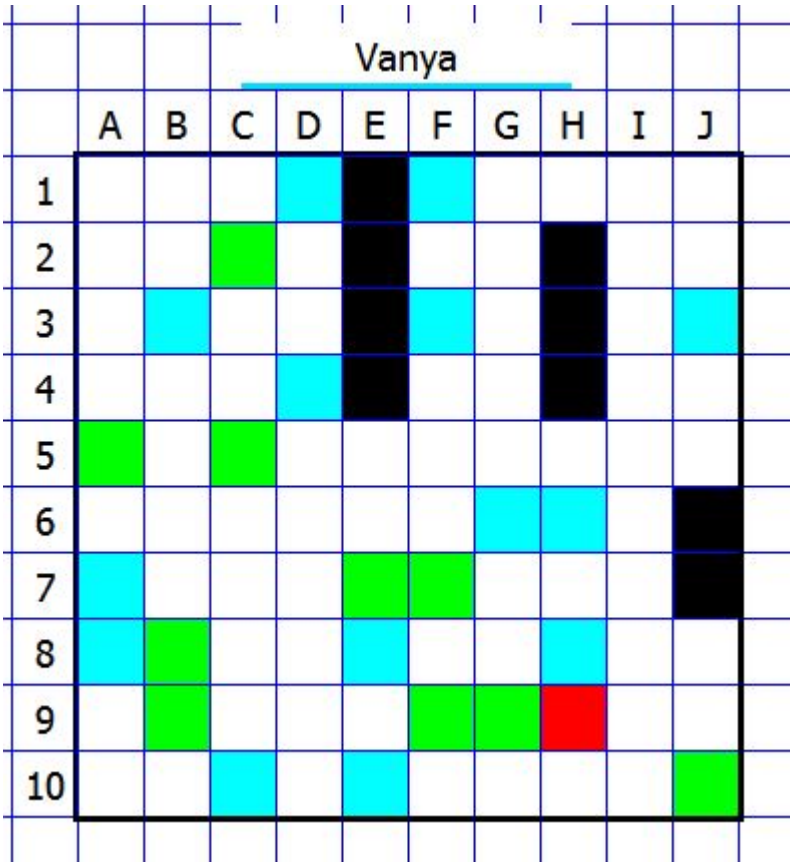

Рис. 2. Первый удачный и второй неудачный этап проведения атаки

На следующем этапе подобным образом ведется поиск второй палубы корабля из тех ячеек поля, которые находятся вокруг первой (всего их 4). Ячейки, на которых также может располагаться самый большой корабль, находятся в более высоком приоритете. При успешном поиске, сортируем

раненые палубы по возрастанию, определяем вертикально или горизонтально находится атакуемый корабль.

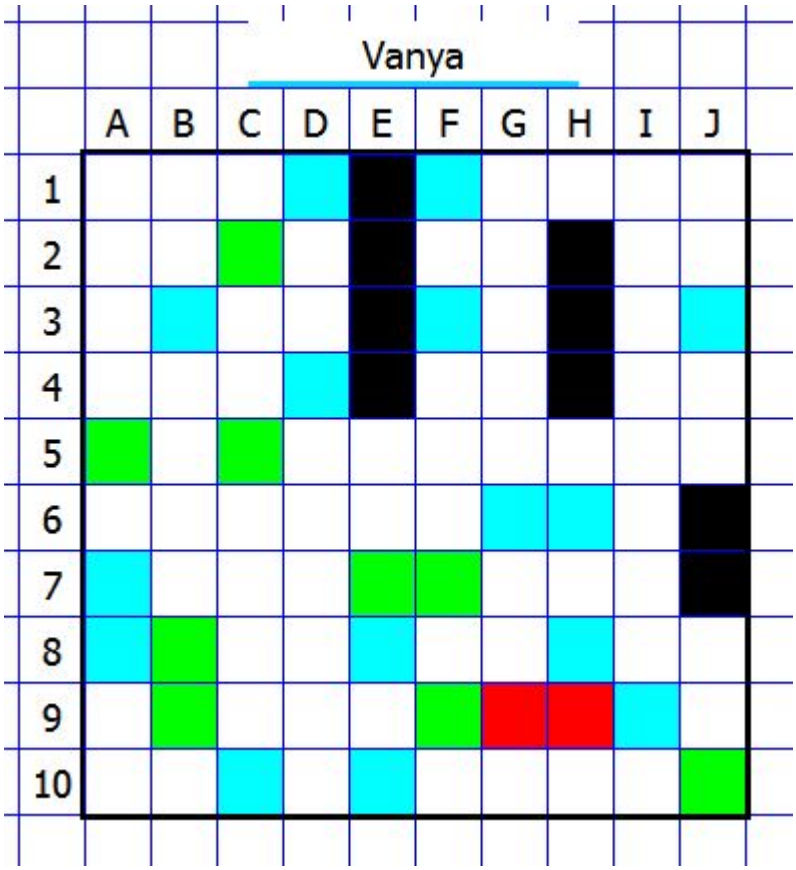

Рис. 3. Второй удачный и третий неудачный этап

На последнем, заключительном, этапе определяем случайное направление для поиска последующих палуб: либо производим атаку впереди первой известной раненой палубы, либо сзади последней. Заключительный этап повторяется до тех пор, пока корабль не будет убит.

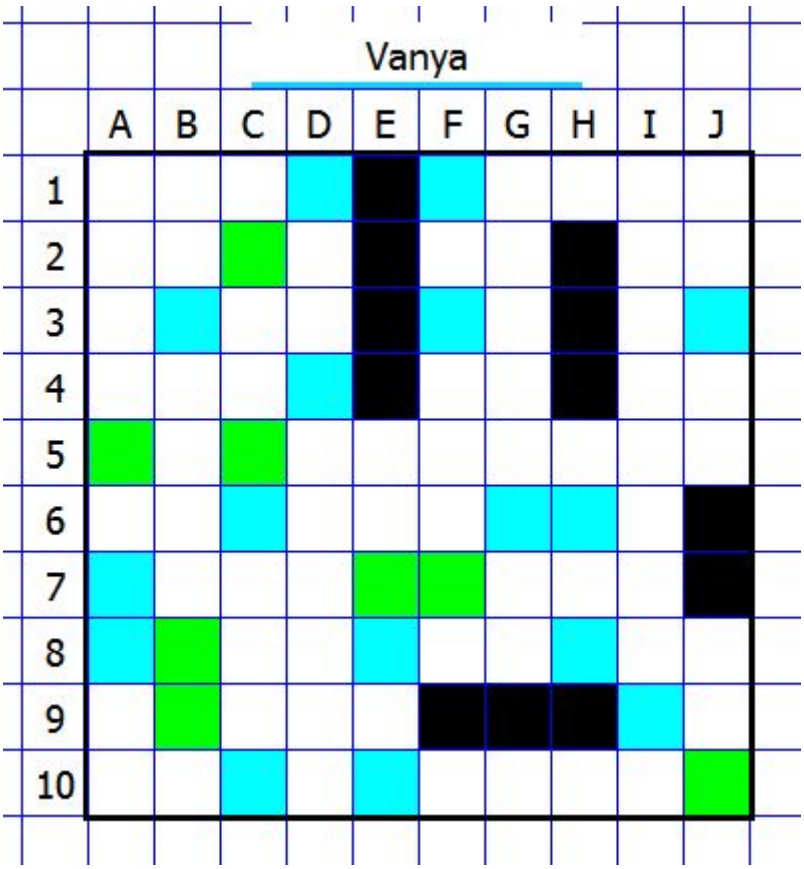

Рис. 4. Заключительный удачный третий этап

Позиция промаха, а также позиции убитого корабля и области вокруг него вычитаются из массива всех допустимых значений для выстрела. Выбранная позиция для выстрела передается в отдельную функцию, реализованную по типу предиката, которая возвращает значение о результате выстрела. Вычитаемая область вокруг убитого корабля вычисляется также в отдельной функции.

Способы победить данного бота в равной степени те же, что и применяются при игре людей друг с другом. Позиции его кораблей также абсолютно случайны. Есть возможность увеличить сложность одиночной игры, не отходя от идеи, как с точки зрения наблюдателя, должен вести себя бот, - реализовать возможность запоминать ранние расстановки пользователя и производить атаку, учитывая высчитанную статистику.

Сетевая игра реализована благодаря классам QTcpSocket и QTcpServer, по протоколу tcp. Особенностью является то, что компьютер в сети может выступать и в роли сервера, и в роли гостя.

При подключении происходит обмен данными: с помощью бинарного оператора «<<» класса QDataStream мы заполняем установленный QByteArray, который хранит информацию побайтово и отправляем информацию второму игроку о эмблеме, имени игрока, позиции его кораблей через QTcpSocket.

Подключение осуществляется через указанный Port и IP. Играть можно как в глобальной сети, так и во внутренней.

СПИСОК ЛИТЕРАТУРЫ

1. Стивен Прата. Язык программирования C++. 2013.
2. Why Does Qt Use Moc for Signals and Slots | Qt5.8 [электронный ресурс] // Qt. 2017. URL: http://doc.qt.io/qt-5/why-moc.html (дата обращения: 22.04.2017).
3. Qt – Product | The IDE [электронный ресурс] // Qt. 2017. URL: https://www.qt.io/ide (дата обращения: 22.04.2017).
4. Макс Шлее. Qt 5.3, профессиональное программирование на C++. 2015.
5. C++11 Language Extensions — General Features, C++ FAQ [электронный ресурс] // Standard C++ Foundation. 2017. URL: https://isocpp.org/wiki/faq/cpp11-language#lambda (дата обращения: 22.04.2017).
6. Qt Wiki. New Signal Slot Syntax – Qt Wiki [электронный ресурс] // Qt Wiki: Here the Qt community has gathered information on Qt over the years. 2017. URL: https://wiki.qt.io/New_Signal_Slot_Syntax (дата обращения: 22.04.2017).